Helsingin yliopisto

Tietojenkäsittelytieteen laitos

Ohjelmistotuotantoprojekti XPerf

Suunnitteludokumentti

Tommi Koivula
Antti Levomäki
Juha Mondolin
Timo Suomela

Versio 1.0.4
24. maaliskuuta 2003

Versiohistoria

Versio	Päivämäärä	Kommentti / muutokset	Tekijä
0.1	26.2.2003	Runko	Juha Mondolin
0.2	5.3.2003	Osajärjestelmät	Tommi Koivula
0.3	10.3.2003	Prolog-luku, lukijan valintajärjestelmä	Juha Mondolin Antti Levomäki
1.0	17.3.2003	asiakkaalle toimitettu versio	Tommi Koivula Juha Mondolin Antti Levomäki Timo Suomela
1.0.3	24.3.2003	Asiakkaan pyytämät korjaukset tehtynä.	Juha Mondolin

Sisällys

1. Johdanto

1.1. Ohjelmiston kuvaus

Toteutettava tuote on laajennus MAISA-ohjelmistoon[1]. MAISA on ohjelmisto, jonka avulla voidaan mitata ohjelmiston erinäisiä laatuominaisuuksia jo suunnitteluvaiheessa. Järjestelmä lukee tiedostosta UML-kaavioita ja valmistaa näiden pohjalta mittareita, jotka kuvaavat suunnitteilla olevan järjestelmän laatua.

Järjestelmään toteutetaan tuki XMI-muodossa oleville UML-kaavioille, mikä mahdollistaa XMI-dokumenttien lukemisen MAISA-työkaluun. Toteutettava lisäosa pystyy lukemaan ainakin ArgoUML-työkalun[2] tuottamia dokumentteja. Lisäosa lukee dokumenteista vain ne tiedot, joita tämänhetkinen MAISA-työkalu tarvitsee, vaikka dokumentit sisältäisivätkin enemmän tietoa. Yksityiskohtainen ohjelmiston määrittely löytyy määrittelydokumentista[5].

1.2. Tärkeimmät vaatimukset

Projektin onnistumisen kannalta kriittisiä vaatimuksia ovat toimivan XMI-tuen toteuttaminen sekä sen integroiminen järjestelmään siten, että myöhemmin myös muita tiedostomuotoja voidaan tukea helposti muuttamatta oleellisesti MAISAn rakennetta. Myös tiedoston lukuoperaation edistyminen pitää ilmaista käyttäjälle.

1.3. Suunnittelusta ja suunnitterurajoituksista

Laajennuksen toteuttamista rajoittavat MAISA-järjestelmän arkkitehtuuri sekä tuettavat standardit XMI [4] ja UML [3].

2. Osajärjestelmien yleiskuvaus

2.1. Toteutettavat osajärjestelmät

MAISA-työkaluun tehtävät lisäominaisuudet jaetaan neljään osajärjestelmään: Lukumoduulin valintajärjestelmä, XMI-lukumoduuli, Prolog-lukumoduuli ja tiedostonluvun käyttöliittymä.

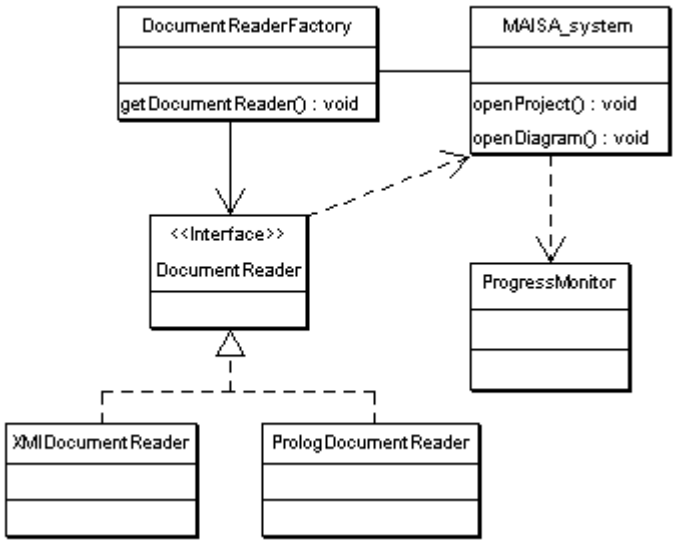
Lukumoduulin valintajärjestelmä tehdään siten, että sen vaikutukset MAISA-työkalun muihin osajärjestelmiin ovat mahdollisimman pienet. Valintajärjestelmä tekee kuitenkin uusien lukumoduulien lisäämisen mahdolliseksi ilman merkittäviä muutoksia MAISA-järjestelmään.

XMI-dokumentin lukuoperaatiot jaetaan kaaviotyyppien mukaisesti omiin luokkiinsa. Sopivan luokan valitsemiseksi käytetään tehdas-suunnittelumallia. Yksityiskohtainen kuvaus löytyy luvusta 3.2.

Prolog-lukumoduuli ei varsinaisesti sisällä uutta toteutusta, vaan siihen kootaan jo olemassaolevat prolog-dokumenttien lukuoperaatiot. Olemassaolevassa järjestelmässä tiedostojen lukeminen tapahtuu *CDiagram*-luokan aliluokissa. Jokainen kaaviotyyppi on oma luokkansa ja hoitaa itse omat lukuoperaationsa. Järjestelmää muutetaan siten, että olemassa olevat Prolog-lukuoperaatiot eriytetään kaavioista ja kerätään yhdeksi osajärjestelmäksi (kts. luku 3.3).

Tiedostonluvun käyttöliittymä toteuttaa dokumenttien lataamiselle asetetut käytettävyysvaatimukset. Sen toiminta ja rakenne selitetään yksityiskohtaisesti luvussa 3.4.

2.2. Toteutettavien osajärjestelmien välinen kommunikointi



Kuva 1 – Osajärjestelmien väliset suhteet

Osajärjestelmien välisiä riippuvuuksia kuvataan kuvassa 1. Lukumoduulin valintajärjestelmän kommunikointi moduuleihin tapahtuu *DocumentReader*-rajapinnan avulla (kts. luku 3.1). Myös tiedostonluvun käyttöliittymä on sidottu moduuleihin pelkästään *DocumentReader*-rajapinnan avulla.

Lukumoduulit eivät kommunikoi lainkaan keskenään ja kommunikointi muiden MAISA-järjestelmän osien kanssa tapahtuu *CProject*-luokan tarjoaman rajapinnan avulla (kts. liite A).

3. Osajärjestelmien kuvaus

3.1. Lukumoduulin valintajärjestelmä

Lukumodulin valintajärjestelmä toteutetaan tehdas-suunnittelumallin mukaisesti. Tehtaalta voidaan tiedostoja avatessa pyytää tiedostoa lukevaa lukumoduulia, joka palautetaan jos sellainen on olemassa.

Tehtaana toimiva *DocumentReaderFactory*-luokka lukee MAISAn käynnistyessä konfiguraatiotiedostosta lukijaluokkien nimet ja luo kustakin luokasta *Class.newInstance()*-metodilla uuden ilmentymän. Tämän jälkeen *DocumentReaderFactory.getReader (java.io.BufferedReader in*)-metodilla voidaan pyytää parametrina annetun tiedoston lukuun sopivan *DocumentReader*-luokan ilmentymää.

Edellytys lukijaluokalle on että se toteuttaa *MAISA.DocumentReader*-rajapinnan. Kun *DocumentReaderFactory*-luokan *getReader (java.io.BufferedReader in*)-metodia kutsutaan, käydään kaikki rekisteröidyt lukijat läpi ja kysytään kunkin kohdalla *MAISA.DocumentReader*-rajapinnassa määritellyllä *canRead(java.io.BufferedReader reader)*-metodilla, osaako lukija tulkata sisällön. Jos lukija osaa käsitellä kyseistä tiedostoa, sen *canRead()*-metodi palauttaa arvon true, jolloin *DocumentReaderFactory.getReader()* palauttaa kyseisen lukijan. Jos yksikään lukija ei osaa käsitellä tiedostoa, *DocumentReaderFactory.getReader()* palauttaa arvon *NULL*.

MAISA.DocumentReaderFactory-luokan metodit:

MAISA.DocumentReader()
MAISA.DocumentReaderFactory.getDocumentReader(java.io.BufferedReader in)

Palauttaa sellaisen *MAISA.DocumentReader*-luokan ilmentymän joka osaa lukea in-parametrina annetun tiedoston tai *NULL*-arvon jos sopivaa lukijaa ei löydy.

Lukumoduulin rajapinta

Edellytys lukijaluokalle on, että se toteuttaa *MAISA.DocumentReader* rajapinnan:

```
public boolean canRead(java.io.BufferedReader in) throws IOException;
public void read(java.io.BufferedReader in, java.io.PrintWriter err,
CProject project) throws IOException, MaisaParseException;
public boolean supportsCancel();
public void cancel();
```

public boolean canRead(java.io.BufferedReader in) throws IOException;

> Metodille annetaan parametrina kahva tutkittavaan syötteeseen, ja jos lukija osaa käsitellä kyseistä tiedostoa, metodikutsu palauttaa totuusarvon *true*, muuten *false*.

public void read(java.io.BufferedReader in, java.io.PrintWriter err, CProject project) throws IOException, MaisaParseException;

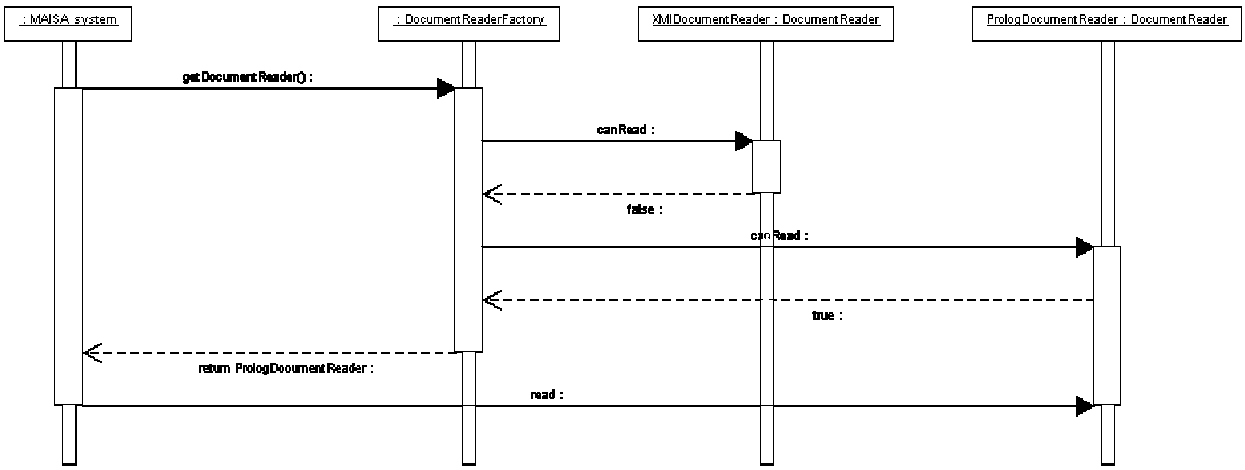
> Kaavion lukumetodin parametri *in* välittää kahvan luettavaan syötteeseen. Parametri *err* välittää kahvan virhesyötteeseen ja parametri *project* välittää viitteen projekti-olioon, jonne tiedot tallennetaan. Metodi heittää kaikki syötteestä peräsin olevat poikkeukset (java.io.IOException) eteenpäin. Jos tiedoston sisältöä ei pystytä jostain syystä jäsentämään, lukijan tulee heittää *MAISA.MaisaParseException*-poikkeus.

public boolean supportsCancel();

Metodi palauttaa arvon *true,* jos lukija tukee lukuoperaation keskeyttämistä, muuten metodi palauttaa arvon *false*.

public void cancel();

Jos lukija tukee sitä, tämä metodi keskeyttää lukuoperaation. Keskeytyksen tukemista voidaan selvittää *supportsCancel()*-metodilla.



Kuva 2 – sekvenssikaavio lukumoduulin valinnasta

3.2. XMI-lukumoduuli

XMI-lukumoduuli toteuttaa XMI-dokumenttien lukemisen MAISA-työkaluun. Tiedot tallennetaan suoraan annetun *CProject*-olion jo olemassaoleviin tietorakenteisiin.

Jotta MAISA-työkalun tietorakenteet pysyvät eheänä, tiedot on luettava tietyssä järjestyksessä. Esimerkiksi kaikki tietorakenteiden oliot on luotava ennen kuin niihin voidaan viitata toisista olioista. Liitteessä B on kuvattu järjestys, jota on noudatettava kaikkien lukumoduulien toiminnassa. Itse XMI-dokumentissa tiedot voivat olla missä tahansa järjestyksessä.

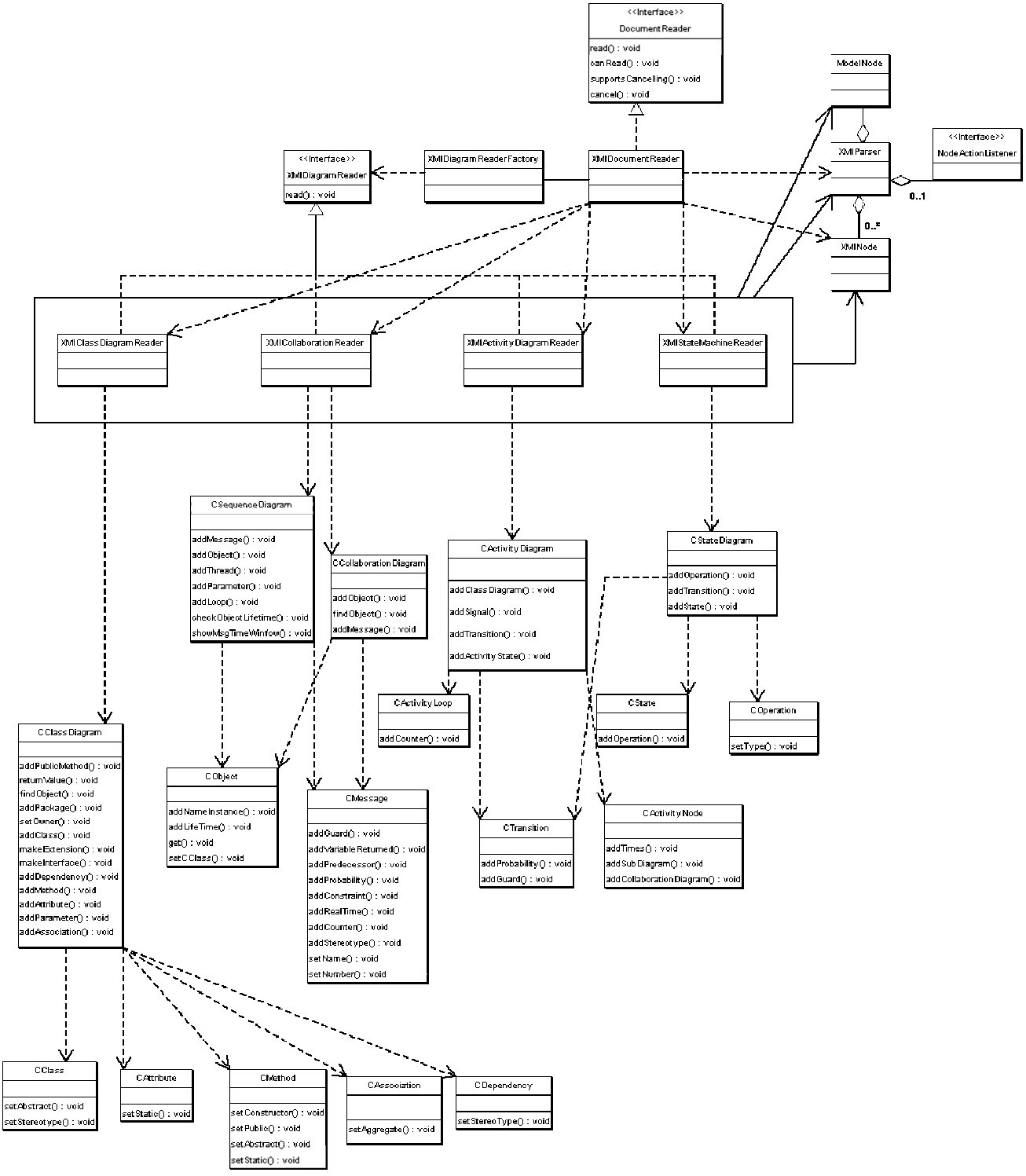
Rajapinnan kuvaus ulkoisiin elementteihin

XMI-luku toteuttaa luvussa 3.1 esitetyn *DocumentReader*-rajapinnan. XMI-lukumoduuli ei sisällä muita rajapintoja muihin osajärjestelmiin.

Osajärjestelmän lukuoperaatioiden lukemat tiedot tallennetaan *CProject*-olioon, jonka viite välitetään lukumoduulille parametrina. *CProject*-luokan jo olemassaolevan rajapinnan avulla päästään käsiksi MAISA-työkalun tarvittaviin tietorakenteisiin. Tarkemmat tiedot käytettävistä luokista ja niiden rajapinnoista jokaisen kaaviotyypin kohdalla erikseen esitellään liitteessä A.

Luokkien kuvaukset

Luokat on kuvattu luokkakaaviona kuvassa 3.



Kuva 3 - XMI-moduulin käyttämät palvelut

XMIDocumentReader

Luokka *XMIDocumentReader* toteuttaa *DocumentReader*-rajapinnan ja toimii näin tiedostonlukijana. Rajapintansa mukaisesti XMIDocumentReader lukee syötteenä annetusta virrasta tarpeelliset tiedot. Luetut tiedot tallennetaan annettuun *CProject*-olioon.

XMIDocumentReader lukee dokumenttiin liittyvät yleiset tiedot ja välittää jokaisen kaaviotyypin elementin (esim. *.Collaboration)* vastaavalle lukijalle. Näitä lukijoita ovat *XMIClassDiagramReader*, *XMICollaborationReader*, *XMIStateMachineReader* ja *XMIActivityDiagramReader*. Vastaava lukijaluokka selvitetään tehdas-mallin avulla. Tehtaana toimii luokka *XMIDiagramReaderFactory*, jolta saadaan aina oikean lukijan ilmentymä. Tämän lukijan avulla suoritetaan itse lukuoperaatiot.

XMIDiagramReaderFactory

XMIDiagramReaderFactory on tehdas, joka tuottaa erilaisille XMI-dokumentin kaaviotyypeille sopivat *XMIDiagramReader*-luokan ilmentymät.

XMIDiagramReaderFactory-luokalla on yksi julkinen metodi: *getXMIDiagramReader(String)*, joka palauttaa *XMIDiagramReader*-olion. Metodille annetaan parametrina elementin nimi, jonka perusteella se valitsee oikean lukijaluokan ja palauttaa olion, joka osaa lukea parametrina annetun elementtityypin.

XMIDiagramReader

XMIDiagramReader on lukijan rajapinta, joka lukee XMI-dokumentin juurielementin (ModelElement) jonkin tietyn lapsielementin sisällön. *XMIDiagramReader*-luokan aliluokista ei tehdä suoraan ilmentymiä, vaan ilmentymät on luotava käyttäen *XMIDiagramReaderFactory*-olion *getXMIDiagramReader*-metodia.

XMIDiagramReader rajapinnassa on vain yksi metodi:

read(TreeWalker walker, CProject project).

Walker-parametri on jäsentimen kulkija dokumentissa, joka osoittaa ko. luettavaan lapsielementtiin. *Project*-parametri on olio, johon luettavat tiedot tallennetaan.

Rajapinnan toteuttavat luokat käyttävät toteutuksiinsa *XMIParser*-jäsennintä, joka kuvataan alla.

XMIParser

XMIParser on XMI-jäsennin, jolle annetaan jäsennettävästä dokumentin osasta hierarkkinen malli. Mallin avulla jäsennin osaa etsiä dokumentista halutut elementit. Malli rakennetaan muodostaen *ModelNode*-solmuista puurakenne. XMI-jäsentimelle annetaan juurisolmu, josta jäsentäminen aloitetaan, ja *TreeWalker*-olio, joka viittaa juurisolmua vastaavaan kohtaan itse dokumentissa.

XMI-jäsentimeen täytyy rekisteröidä jokin *NodeActionListener*-olio, jonka *actionEvent*-metodia kutsutaan jokaiselle dokumentista löytyvälle mallin solmulle. *ActionEvent*-metodissa voidaan kutsun aiheuttaneesta elementistä lukea halutut tiedot.

XMI-jäsennin käyttää dokumentin läpikäymiseen DOM Level 2 jäsentimen *TreeWalker*-oliota ja *NodeFilter*-ilmentymää. *TreeWalker*-olion avulla liikutaan dokumentissa juurisolmusta lehtisolmuja kohden. *NodeFilter*-olio rajaa pois ei-halutut elementit dokumentista. *XMIParser*-luokka toteuttaa itse *NodeFilter*-rajapinnan.

Kun malli on luotu ja jäsennin alustettu, tehdään jäsentäminen *parse*-metodilla. Tällöin dokumentti luetaan ja sieltä etsitään mallin mukaiset elementit. Jos jäsennyksessä kohdataan virheitä, heitetään poikkeus *MaisaParseException*.

Luokka sisältää myös staattiset metodit UML-laajennusten (tagged values) lukemiseen:

```
public static String getTaggedValueStr( TreeWalker parent, String keyName );
public static long getTaggedValueLong( TreeWalker parent, String keyName );
public static float getTaggedValueFloat( TreeWalker parent, String keyName );
public static boolean getTaggedValueBoolean(
                TreeWalker parent, String keyName
);
```

Ym. metodien avulla voidaan lukea dokumentin tietyn elementin (parametri *parent*) sisältämän UML-laajennus (parametri *keyName*) ja muuntaa se suoraan haluttuun muotoon. Jos arvoa ei löydy dokumentista tai muuntaminen haluttuun muotoon ei onnistu, heitetään *MaisaParseException*–poikkeus.

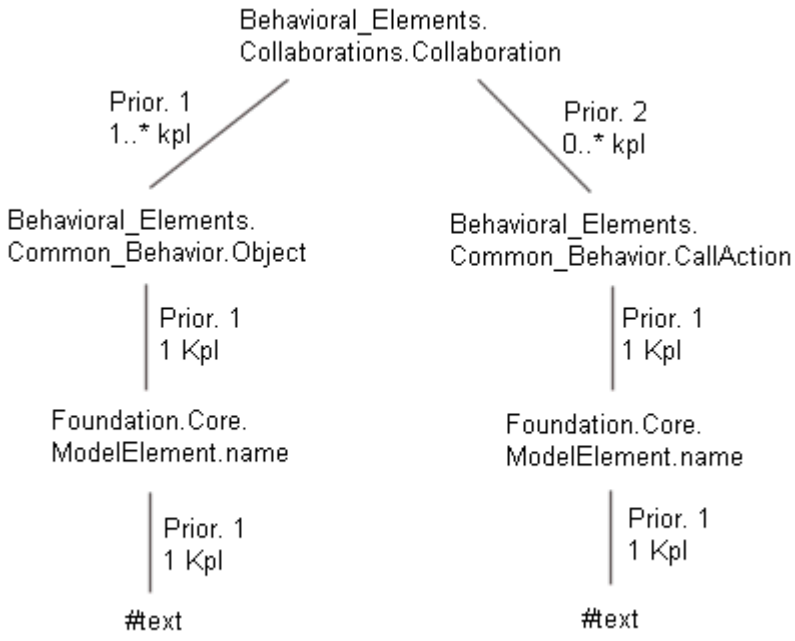
ModelNode

Luokka *ModelNode* on XMI-dokumentin jäsentämiseen tarvittava solmu, jolla hierarkkinen malli halutuista elementeistä rakennetaan. *ModelNode*-luokalla on metodi *addChild*, jonka avulla solmuista voidaan muodostaa puurakenne.

Mallisolmu koostuu nimestä, elementin nimestä, prioriteetista ja lapsista. Solmun nimi tulee olla yksikäsitteinen, jotta solmu voidaan tunnistaa sen perusteella. Jokaisesta lapsisolmusta on myös määriteltävä, montako niitä voi ko. elementissä olla (esim. 1..*).

Solmut, joiden prioriteettinumero on pieni, luetaan aina ennen isompinumeroisia sisarsolmuja. Prioriteetti koskee siis vain sisarsolmuja eli solmuja, joilla on sama äitisolmu. Jos prioriteettinumeroa ei määritellä, solmu luetaan viimeisenä ja saman numeroiset solmut luetaan määrittelemättömässä järjestyksessä.

Esimerkki *ModelNode*-olioilla tehdystä mallista on kuvassa 4.



Kuva 4 - Osa yhteistyökaavion lukemiseen käytettävästä mallista.

NodeActionListener

NodeActionListener rekisteröidään johonkin *XMIParser*-olioon. Aina kun jäsennin löytää mallinsa mukaisen elementin dokumentista, se kutsuu *NodeActionListener*-luokan *actionEvent*-metodia. Metodi saa parametrinaan *XMINode*-olion, joka kuvaa toiminnon aiheuttavaa elementtiä dokumentissa.

XMINode

XMINode kuvaa XMI-dokumentin jäsennyksen solmua. *XMINode*-luokka on myös wrapper-luokka *dom.Node*-luokalle. *XMINode* sisältää kaikki *dom.Node*-luokan metodit, joilla voidaan tarkastella solmun tilaa. Myös metodit solmun attribuuttien hakemiseksi löytyvät *XMINode*-luokasta. Dokumentissa liikkumiseen tarkoitettuja metodeja *XMINode*-

luokasta ei löydy. *XMINode*–oliolla on myös nimi, joka on sama kuin vastaavan mallin solmun nimi (kts. luokka *ModelNode*).

Jäsennin luo *XMINode-luokan* ilmentymän aina kun löytää dokumentista mallissa olevan solmun. Tällöin luotu olio välitetään *NodeActionListener*-oliolle.

XMIClassDiagramReader

XMIClassDiagramReader toteuttaa *XMIDiagramReader*-rajapinnan. *XMIClassDiagramReader* lukee XMI-dokumentista luokkakaavion. XMI-dokumentissa luokkakaavion kaikki osat sijaitsevat dokumentin juurielementissä (*ModelElement*), joten käytännössä luokan *read()*-metodi lukee vain jonkin luokkakaavion osan ja metodia on kutsuttava jokaiselle luokkakaavioon kuuluvalle juurielementin lapselle erikseen.

Toteutukseen käytetään *XMIParser*-luokkaa jäsentimenä. Mikäli jäsennyksessä tapahtuu virhe, heitetään poikkeus *MaisaParseException*.

XMICollaborationReader

XMICollaborationReader toteuttaa *XMIDiagramReader*-rajapinnan. *XMICollaborationReader* lukee XMI-dokumentista sekä yhteistyökaavion että sekvenssikaavion, koska ne molemmat sijaitsevat dokumentin samassa elementissä (*.Collaborations.Collaboration*).

Toteutukseen käytetään *XMIParser*-luokkaa jäsentimenä. Mikäli jäsennyksessä tapahtuu virhe, heitetään poikkeus *MaisaParseException*.

XMIStateMachineReader

XMIStateMachineReader toteuttaa *XMIDiagramReader*-rajapinnan ja lukee tilakaavion.Toteutukseen käytetään *XMIParser*-luokkaa jäsentimenä.

Mikäli jäsennyksessä tapahtuu virhe, heitetään poikkeus *MaisaParseException*.

XMIActivityDiagramReader

XMIActivityDiagramReader toteuttaa XMIDiagramReader-rajapinnan ja lukee aktiviteettikaavion. Toteutukseen käytetään *XMIParser*-luokkaa jäsentimenä.

Mikäli jäsennyksessä tapahtuu virhe, heitetään poikkeus *MaisaParseException*

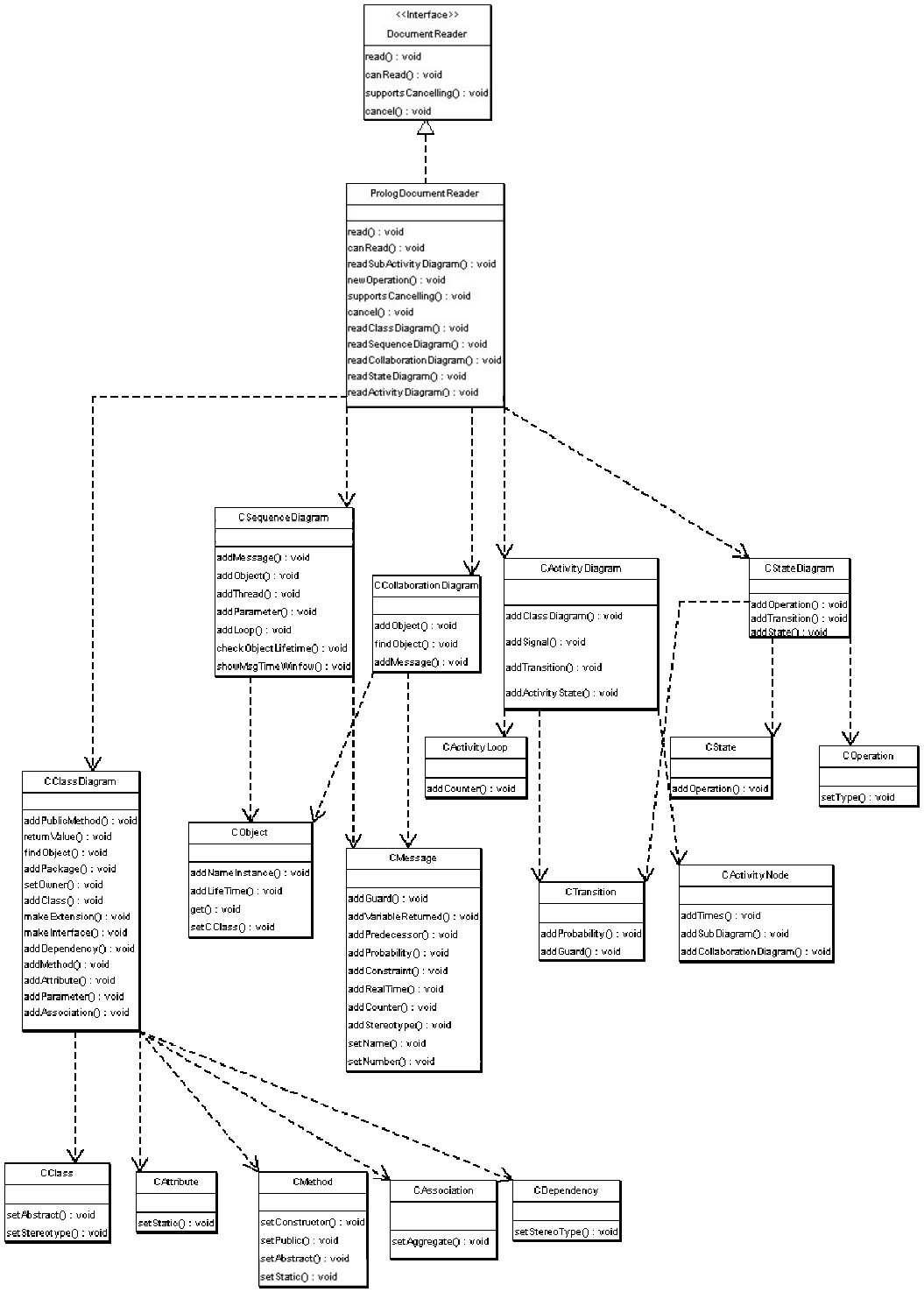
3.3. Prolog-lukumoduuli

Prolog-muodossa olevien kaavioiden lukeminen MAISAan tapahtuu samoilla periaatteilla kuin ennenkin, mutta lukuoperaatiot on irrotettu kaavioita kuvaavista luokista ja siirretty yhteen prolog-tiedostojen lukemisesta vastaavaan luokkaan. Prolog-lukumoduulin sijoittumista MAISAan on kuvauttu kuvassa 5.

Prolog-lukija (*PrologDocumentReader*) lukee prolog-muodossa olevien kaavioiden kuvaukset tiedostoista MAISAn tietorakenteisiin. Lukija toteuttaa *DocumentReader*-rajapinnan. Sen *read()*-metodille annetaan parametrina se *Cproject*-olio, johon luettavat kaaviot halutaan liittää. Itse metodissa kutsutaan luettavan kaavion tyyppiä vastaavaa lukumetodia(esim. *readClassDiagram*), jolle annetaan parametrina viite luettavaa kaaviota vastaavaan, *CDiagram*-luokan toteuttavasta luokasta(esim. *CClassDiagram*) luotuun, olioon. Tähän olioon metodi tallentaa kaavioon liittyvät tiedot käyttäen ko. luokan julkisia metodeita.

Lukumetodien toteutus on lähes identtinen nykyisten *CDiagram*-luokasta perivien luokkien *read()*-metodien kanssa. Tietojen tallettamiseen kutakin kaaviotyyppiä vastaavaan olioon käytetään ko. luokkien julkisia metodeja. Nämä metodit on lueteltu kaaviotyypeittäin liitteessä A.

PrologDocumentReader –luokan varsinainen lukumetodi *read()* vastaa toiminnaltaan nykyisen *CProject*-luokan lukumetodia *readProject()*. Metodi käyttää joitakin *CProject*-luokan sisäisiä muuttujia, joita on ennen käsitelty suoraan. Nyt tähän tarkoitukseen käytetään CP*roject*-luokan julkista *addDiagram()*-metodia joka osaa päivittää CProjectin sisäiset muuttujat.



Kuva 5 - Prolog-lukumoduulin käyttämät palvelut

3.4. Tiedostonluvun käyttöliittymä

Edistysindikaattori

Kun käyttäjä on valinnut tiedoston ladattavaksi, käynnistyy taustalle siirtymäajastin. Jos ajastin laukeaa ennen kuin tiedosto on kokonaan ladattu, niin käyttäjälle näytetään pienessä ikkunassa edestakaisin liikkuva indikaattori. Jos automaattisesti valittu tiedostonlukija tukee keskeyttämistä, ikkunassa näkyy käyttäjälle nappi, jota painamalla tiedoston latauksen voi keskeyttää.

Muutokset kohdistuvat *MAISA_system*-luokan *openProject()* ja *openDiagram()*-metodeihin. Indikaattori toteutetaan käyttämällä javax.swing.ProgressMonitor-luokan ilmentymää.

4. Muutoksia olemassa olevaan järjestelmään

4.1. Yksittäisen kaaviotiedoston lukumekanismi

Olemassa olevassa järjestelmässä yksittäinen kaaviotiedosto voidaan lukea siten, että se lukemisen jälkeen on itsenäinen, eli ei kuulu mihinkään projektiin. XMI-dokumentti voi sisältää useamman kaavion, jotenka tehtävien muutosten jälkeen yksittäisen kaaviotiedoston sisältö luetaan aina uuden projektin alaiseksi.

Muutokset kohdistuvat *MAISA_system*-luokan *openProject()* ja *openDiagram()*-metodeihin.

4.2. Kuva- ja laskentatulostiedostojen lataus

Olemassa olevassa järjestelmässä kuva- ja laskentatulostiedostot on nimetty sen tiedoston mukaan, joka sisältää kaavion, johon kyseinen informaatio liittyy. Prolog-formaatissa olevien tiedostojen lukemisen jälkeen käydään tiedostot sisältävä hakemisto läpi ja tarkistetaan kunkin JPEG tai GIF-formaatissa olevan tiedoston kohdalla, onko luettu Prolog-tiedostoa, jonka nimi poikkeaa ainoastaan pistettä seuraavan päätteen kohdalla. Jos tällainen kaavio löytyy, niin kuvatiedosto liitetään siihen. Samaan tapaan myös laskentatulostiedostot on kytketty niitä vastaaviin kaaviotiedostoihin.

XMI-dokumentti voi sisältää useamman kaavion, mistä johtuen kuva- ja laskentatulostiedostot voidaan tehtävien muutosten jälkeen nimetä tyyliin <kaavio_tiedoston_nimi>-<kaavion_nimi>.<tiedoston_pääte>. Yllä esiteltyyn tapaan nimetty tiedosto liitetään <kaavio_tiedosto_nimi>.<formaattipääte>-nimisestä tiedostosta luettuun <kaavion_nimi>-nimiseen kaavioon..

Muutokset kohdistuvat *MAISA_system*-luokan *openProject()* ja *openDiagram()*-metodeihin.

4.3. Poikkeusten käsittely

Nykyisellään MAISA:n tiedostonluvussa heitetään poikkeukset sekä suoraan että käytetään *MaisaStructure.throwError()*-metodia. Toteutettavissa lukujärjestelmissä (XML ja Prolog) *MaisaStructure.throwError()*-metodia ei enää käytetä.

5. Yhteenveto

MAISA-järjestelmään tehtävät uudet ominaisuudet on jaettu neljään osajärjestelmään: Lukumoduulin valintajärjestelmä, XMI-lukumoduuli, Prolog-lukumoduuli sekä

tiedostoluvun käyttöliittymä. Näiden arkkitehtuurit ja rajapinnat muihin osajärjestelmiin on tässä dokumentissa yksityiskohtaisesti esitetty.

Tämän dokumentin sisältö vastaa vaatimusten, joita toteutettavalle ohjelmistolle on määrittelydokumentissa esitetty, toteuttamiseen tarvittavia tietoja. Dokumentti esittelee siis toteutettavan ohjelmiston osan arkkitehtuurin tarkkuudella, jolla se voidaan käytännössä toteuttaa.

Lähteet

[1] MAISA, Metrics for Analysis and Improvement of Software Architectures http://www.cs.helsinki.fi/group/maisa/

[2] ArgoUML, Copyright (C) 1996-2002 The Regents of the University of California , [http://argouml.tigris.org/]

[3] Unified Modelling Language (UML) version 1.3 spesifikaatio, Copyright (C) 1997-2003 Object Management Group, Inc., [http://www.omg.org/cgi-bin/doc?formal/00-03-01].

[4] XML Metadata Interchange (XMI) version 1.0, Copyright © 1997-2003 Object Management Group, Inc., [http://www.omg.org/cgi-bin/doc?formal/00-06-01]

[5] Xperf –ohjelmistotuotantoprojektin määrittelydokumentti, [http://www.cs.helsinki.fi/group/maisa/xperf/doc/Maarittelydokumentti_1.1.6.PDF]

Liite A - Rajapinnat MAISA-järjestelmän tietorakenteisiin

Aktiviteettikaavio

Aktiviteettikaavion lukuoperaatio luo seuraavien luokkien ilmentymiä:

CactivityNode	(Geneerinen solmu, voi olla joko ActivityLoop tai ActivityState)
CactivityLoop	(Silmukan alkusolmu)
CTransition	(siirtymä tilojen välillä)
CActivityDiagram	((ali)aktiviteettikaavio)

Tietojen tallentamiseen käytetään seuraavia metodeja:

CActivityDiagram	
Metodi	**Selitys**
addSignal(String oid,String name,String type)	Lisää signaalin.
addTransition(String oid,CActivityNode from,CActivityNode to)	Lisää siirtymän.
addClassDiagram(String oid)	Lisää sen luokkakaavion tunnuksen, johon tämä aktiviteettikaavio liittyy.
addActivityState(String oid,String name,int type)	Lisää aktiviteettitilan
read(BufferedReader f,PrintWriter stdout,int lines)	Lukee alikaavion.
FindObject(String getWhat)	Palauttaa annettua tekstuaalista nimeä vastaan elementin.

CTransition	
Metodi	**Selitys**
addProbability(String newProb)	Lisää todennäköisyyden.
addGuard(String guardToBeAdded)	Lisää guardin siirtymään.

CActivityNode	
Metodi	**Selitys**
addSubDiagram(CActivityDiagram newSub)	Lisää alidiagrammin.
addTimes((String newTime1, String newTime2, String newTime3)	Lisää suoritusaikatietoa (minimi, keskimääräinen ja maksimi)
addCollaborationDiagram(String oid)	Ei toteutusta

CActivityLoop	
Metodi	**Selitys**
addCounter(String newCount1, String newCount2, String newCount3)	Lisää countterin.

Tilakaavio

Tilakaavion lukuoperaatio luo seuraavien luokkien ilmentymiä:

CState (Tila)
CTransition (Tilojen välinen siirtymä)
COperation (Tilassa suoritettava toiminto)

Tietojen tallentamiseen käytetään seuraavia metodeja:

CStateDiagram	
Metodi	**Selitys**
addState(int oid, String name)	Lisää tilakaavioon uuden tilan tunnuksella 'oid' ja nimellä 'name'.
addOperation(int oid, String name)	Lisää tilakaavioon uuden toiminnon tunnuksella 'oid' ja nimellä 'name'.
addTransition(CState from, CState to)	Lisää tilakaavioon uuden siirtymän tilasta 'from' tilaan 'to'.

CState	
Metodi	**Selitys**
addOperation(COperation operation)	Liittää tilakaavion tilaan toiminnon 'operation'.

Coperation	
Metodi	**Selitys**
setType(String string)	Asettaa toiminnon tyypin ("Exit", "Do"); "Entry" ei aseteta.

CTransition	
Metodi	**Selitys**
addGuard(String name)	Liitää tilakaavion siirtymään ehdon 'name'.

Luokassa CDiagram, josta CStateDiagram perii on määritelty muuttuja 'objects' näkyvyydellä 'protected'. Muuttujaa käytetään suoraan, mutta se voidaan korvata olemassa olevalla, CDiagram luokass määritetyllä 'findObject(String name)' metodilla.

Luokkakaavio

CClassDiagram	
Metodi	**Selitys**
addPublicMethod()	Lisää julkisten metodien lukumäärä.
returnValue(CMethod m, String s)	Asettaa metodin *m* paluuarvoksi tyypin *s*.
addPackage(String oid, String name)	Lisää pakkauksen, jonka tunnus on *oid* ja nimi *name*.
setOwner(CClass owned, CPackage owner)	Asettaa luokan *owned* kuuluvaksi *owner* pakkaukseen.
setOwner(CPackage owned, CPackage owner)	Asettaa pakkauksen *owned* kuuluvaksi *owner* pakkaukseen
addClass(String oid, String name)	Lisää luokan.
makeExtension(CClass sub, CClass super)	Asettaa luokan *sub* periytyväksi luokasta *super*.
makeInterface(CClass implementor, CClass implementee)	Asettaa luokan *implementor* rajapinnan *implementee* ilmentymäksi.
addDependency(String oid, CClass first, CClass second)	Asettaa riippuvuuden, jonka tunnus on *oid* ja luokat *first* ja *second.*
addMethod(CClass owner, String oid, String name)	Lisää metodin, jonka tunnus on *oid* ja nimi *name*, luokkaan *owner.*
addAttribute(CClass owner, String oid, String name, String type)	Lisää attribuutin, jonka tunnus on *oid*; nimi *name* ja tyyppi *type*, luokkaan *owner.*
addParameter(CMethod method, String oid, String name, String type)	Lisää metodiin *method* parametrin, jonka tunnus on *oid*; nimi *name* ja tyyppi *type*.
addAssociation(String oid, CClass c1, String const1, CClass c2, String const2)	Lisää uuden assosiaation luokkien *c1* ja *c2* välille.
findObject(String getWhat)	Palauttaa annettua tekstuaalista nimeä vastaan elementin.

CClass	
Metodi	**Selitys**
setAbstract()	Asettaa luokan abstraktiksi.
setStereotype(String value)	Asettaa luokan stereotyypiksi *valuen*.

CAttribute	
Metodi	**Selitys**
setStatic()	Asettaa attribuutin staattiseksi.

Cmethod	
Metodi	**Selitys**
setConstructor()	Asettaa metodin luokkansa konstruktoriksi.
setPublic()	Asettaa metodin julkiseksi.
SetAbstract()	Asettaa metodin abstraktiksi.
setStatic()	Asettaa metodin staattiseksi

CDependency	
Metodi	**Selitys**
boolean setStereotype(String newStereotype)	Asettaa riippuvuuden stereotyypiksi *newStereotypen*. Sallittuja stereotyyppejä ovat *”creates”* ja *”refers”*.

CAssociation	
Metodi	**Selitys**
setAggregate(boolean isComposite, CClass theComposite)	Asettaa assosiaation aggregaatin. Jos *isComposite* on *true*, on kysessä kompositio.

Sekvenssikaavio

CSequenceDiagram	
Metodi	**Selitys**
addObject(String oid, String name)	Lisää olion.
addThread(String oid, String name)	Lisää säikeen.
addParameter(String oidMes, String oidElem, int order, CSequenceParameter.ELEMENT)	Lisää viestiin *oidMes* muuttujaparametrin, jonka tunnus on *oidElem* ja järjestysluku *order*.
addParameter(String oidMes, String oidElem, int order, CSequenceParameter.VARIABLE)	Lisää viestiin *oidMes* olioparametrin, jonka tunnus on *oidElem* ja järjestysluku *order*.
addMessage(String oid, String name, CElement from, CElement to, int number, CMessage.NORMAL_MESSAGE)	Lisää normaalin viestin oliolta *from* oliolle *to*, jonka järjestysluku on *number*.
addMessage(String oid, String name, CElement from, CElement to, int number, CMessage.STEREOTYPE_MESSAGE)	Lisää sterotyyppiviestin oliolta *from* oliolle *to*, jonka järjestysluku on *number*.
addLoop(String oid1, String oid2, int order_first, int order_last, int min, int max, int typical)	Lisää silmukan, jonka tunnukset ovat *oid1*ja *oid2*, viestistä *order_first* viestiin *order_last*. Silmukan suorituskerrat kerrotaan parametreilla *min*,*max*,*typical*.
checkObjectLifetimes()	Tutkii ovatko olion eliniät asetettu.

showMsgTimeWindow()	Näyttää viestin eliniän -ikkunan.
findObject(String getWhat)	Palauttaa annettua tekstuaalista nimeä vastaan elementin.

Cobject	
Metodi	**Selitys**
addNameInstance(String name)	Asettaa ilmentymän nimen.
addLifeTime(int order_first, int order_last)	Asettaa olion eliniän kestämään viestistä *order_first* viestiin *order_last*.

Cmessage	
Metodi	**Selitys**
addGuard(String guard)	Asettaa ehtolausekkeen.
addVariableReturned(String name)	Asettaa viestin paluuarvoksi *name* nimisen olion.
addConstraint(int time)	Asettaa viestin suoritusajaksi *time*.
addRealTime(float lower_time, float typical_time, float upper_time)	Asettaa viestin suoritusajan pienimmillään, suurimmillaan ja keskimäärin.

Yhteistyökaavio

Yhteistyökaavion lukuoperaatio luo seuraavien luokkien ilmentymiä:

CMessage

CObject

Tallentamiseen käytetään seuraavia metodeja:

CCollaborationDiagram	
Metodi	**Selitys**
addObject(String oid, String name)	lisää kaavioon uuden objektin tunnuksella ‘oid’ ja nimellä ‘name’
findObject(String getWhat)	Palauttaa annettua tekstuaalista nimeä vastaan elementin.
addMessage(String oid, String from, String to)	Lisää kaavioon uuden viestin tunnuksella oid, objektilta from objektille to

Cobject	
Metodi	**Selitys**
get()	
setCClass(String classOid)	asettaa objektin tyypiksi luokan classOid

Cmessage	
Metodi	**Selitys**
addGuard(String aguard)	Asettaa viestille guard-lauseen
addProbability(String aprobability)	Asettaa viestille todennäköisyyden.
AddPredecessor(int anumber)	asettaa viestin edeltäjän *ei tee mitään*
addStereotype(String astereotype)	asettaa viestin stereotyypin
addCounter(String newCount1, String newCount2, String newCount3)	asettaa viestille laskurin. newCount1 on pienin mahdollinen silmukoiden suorituskerta, newCount2 keskimääräinen ja newCount3 suurin mahdollinen.
setName(String newName)	asetaa viestille nimen
SetNumber(String newNumber)	asetaa viestille numeron

Liite B - Order of read operations for MAISA-system

To maintain the integrity of data structures of MAISA-system, all read operations must be done in certain order. The main reason for this is that objects must be created before they can be referenced.

Here is a list for right order of read operations grouped by diagram type. Read operations with smaller numbers must be read first and when there are several operations with the same order number, they can be read in any order.

Order of project data and different diagrams

1. General project data
2. Class diagrams
3. State diagrams, activity diagrams, collaboration diagrams, sequence diagrams
4. Other files (for example image files)

Activity diagram

1. Activitystate, activitynode, activityloop
2. Transitions
3. Guard, decision, probability, time, counter, subdiagram, collaboration diagram

State diagram

1. State
2. Transition
3. Operation, Guard

Class diagram

1. Packages **-** packages that contains other packages, must be read before packages they are containing
2. Classes
3. Operations, attributes, inheritances, depencies, aggregations, compositions, interfaces
4. Parameters of an operation

   Properties of a class: is abstract, stereotype

   Properties of an operation: is constructor, visibility, is abstract, is static, return value

   Properties of an attribute: is static

   Properties of a dependency: stereotype

Sequence diagram

1. Objects, threads
2. Messages

In a XMI-document messages are divided into three separate elements that must be read in the following order:

1. *.Link*
2. *.Stimulus*
3. *.CallAction*

3. Properties of a message:

   message guard, return value, execution time, real execution times

   Parameters of a message

   Properties of an object: object lifetime

   Message loops

Collaboration diagram

1. Roles (*Behavioral_Elements.Collaborations.ClassifierRole*)

   First id and name, then the rest

2. Messages (*Behavioral_Elements.Collaborations.Message*)

   First id, the sending- and receiving role, then the rest